ALADDIN: Asymmetric Centralized Training for Distributed Deep Learning

Yunyong Ko
Hanyang University
Seoul, Republic of Korea
koyunyong@hanyang.ac.kr

Kibong Choi
Hanyang University
Seoul, Republic of Korea
rlqhd26@hanyang.ac.kr

Hyunseung Jei
SK Telecom
Seoul, Republic of Korea
hsjei@sk.com

Dongwon Lee
The Pennsylvania State University
University Park, PA, USA
dongwon@psu.edu

Sang-Wook Kim*
Hanyang University
Seoul, Republic of Korea
wook@hanyang.ac.kr

ABSTRACT

To speed up the training of massive deep neural network (DNN) models, distributed training has been widely studied. In general, a centralized training, a type of distributed training, suffers from the communication bottleneck between a parameter server (PS) and workers. On the other hand, a decentralized training suffers from increased parameter variance among workers that causes slower model convergence. Addressing this dilemma, in this work, we propose a novel centralized training algorithm, ALADDIN, employing "asymmetric" communication between PS and workers for the PS bottleneck problem and novel updating strategies for both local and global parameters to mitigate the increased variance problem. Through a convergence analysis, we show that the convergence rate of ALADDIN is $O(\frac{1}{\sqrt{nk}})$ on the non-convex problem, where n is the number of workers and k is the number of training iterations. The empirical evaluation using ResNet-50 and VGG-16 models demonstrates that (1) ALADDIN shows significantly better training throughput with up to 191% and 34% improvement compared to a synchronous algorithm and the state-of-the-art decentralized algorithm, respectively, (2) models trained by ALADDIN converge to the accuracies, comparable to those of the synchronous algorithm, within the shortest time, and (3) the convergence of ALADDIN is robust under various heterogeneous environments.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

Distributed deep learning, centralized training, heterogeneous systems

*Corresponding author

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482412

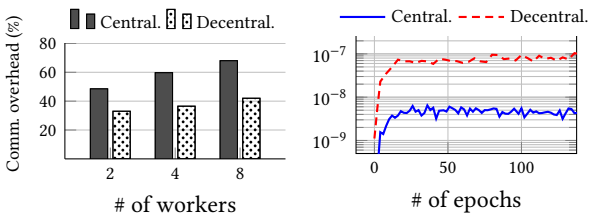
ACM Reference Format:
Yunyong Ko, Kibong Choi, Hyunseung Jei, Dongwon Lee, and Sang-Wook Kim. 2021. ALADDIN: Asymmetric Centralized Training for Distributed Deep Learning. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482412

1 INTRODUCTION

The recent success of deep learning in various fields is underlied by large models and datasets. Though the great advance of a computational accelerator such as GPU has dramatically increased the training speed, training a large model with a large amount of data still requires much time. Thus, several deep learning (DL) frameworks such as Tensorflow and PyTorch train several popular models in advance and provide the pre-trained models to users. However, these pre-trained models are not likely appropriate for the application that users want to apply to. Therefore, it is still important to train a large model from scratch.

To efficiently train large deep neural network (DNN) models such as ResNet-50 [11], VGG-16 [28], BERT [7], and GPT-3 [4] with millions to billions of parameters, *distributed training* has been widely studied [1, 3, 12, 14, 16, 20–22, 24, 25, 30, 36–38]. In distributed training with data parallelism, each worker node trains its local model based on its local data, and then the training results (gradients or parameters) at all workers are aggregated through network communication. Such an aggregation can occur at a dedicated parameter server (PS) (i.e., *Centralized*) or at each worker node via peer-to-peer communication (i.e., *Decentralized*). On the other hand, the communication for the aggregation can be performed either *Synchronously* or *Asynchronously*. Therefore, using these two dimensions, existing distributed training algorithms can be classified into four quadrants as shown in Table 1.

While synchronous training [9, 39], in general, has a good convergence rate thanks to its parameter synchronization across all workers, its synchronization overhead can be significant (especially in heterogeneous environments) [8, 35]. On the other hand, asynchronous training, especially centralized one [12, 25, 30, 36, 37, 40], aims to reduce the synchronization overhead by employing asynchronous communication between PS and workers, but suffers from the problem of PS being a bottleneck. In general, centralized algorithms have larger communication overhead than decentralized



(a) Communication overhead (b) Parameter variance

Figure 1: Communication overhead and parameter variance (Y-axis) of a centralized algorithm (Hogwild) [25] and a decentralized algorithm (SGP) [1].

ones [24]. Decentralized training [1, 2, 21, 22, 31] aggregates the training results of workers via peer-to-peer communication, avoiding the bottleneck problem. However, it becomes more difficult to aggregate the training results via peer-to-peer communication as the size of model parameters and the number of workers increase. It can result in the increased parameter variance among workers, leading to delayed global model convergence.

These contrasting pros and cons of asynchronous centralized and decentralized training are well demonstrated in our preliminary evaluation (Figure 1). Figure 1(a) shows that centralized training incurs more communication overhead while Figure 1(b) shows that decentralized training has larger parameter variance. Motivated from this demonstration, in this work, we aim to *improve asynchronous training, by addressing both problems of large communication overhead and high parameter variance together*. To this end, we can consider two directions: (D1) to reduce the communication overhead in centralized training, and (D2) to reduce the parameter variance among workers in decentralized training. Between the two directions, we excluded (D2) because it is not easy to achieve (D2) efficiently in decentralized training, where global parameters can be obtained only by averaging the parameters of all workers via communication because they do not physically exist.

To achieve (D1), first of all, we note that "symmetric" communication between PS and workers is a critical factor that causes a large communication overhead in centralized training. In the symmetric communication, each worker sends its training result to PS and *has to wait for the updated parameters from PS*. By this symmetricity, when PS becomes a bottleneck, workers are unable to begin the next iteration, waiting for PS. Therefore, by lifting up this symmetricity, we propose a novel approach to centralized data parallelism, **A**symmetric centra**L**ized tr**A**ining for **D**istributed **D**eep learn**IN**g (**ALADDIN**) that employs (1) "asymmetric" communication between PS and workers so that workers begin the next iteration without waiting for the updated global parameters from PS, and (2) novel updating strategies for both local and global parameters to mitigate the problem of increased parameter variance.

In addition, we show that the model trained by ALADDIN converges at the rate of $O(\frac{1}{\sqrt{nk}})$ on non-convex optimization, where n is the number of workers and k is the number of training iterations. Note that this rate is consistent with those of SGD and existing state-of-the-art works [1, 22]. Through comprehensive experiments, we demonstrate that (1) ALADDIN provides training throughput up to 191% and 34% higher than a synchronous decentralized algorithm (AR-SGD) and a state-of-the-art asynchronous decentralized algorithm (SGP), respectively, (2) models trained by ALADDIN converge to the accuracies, comparable to those of AR-SGD within the shortest time, and (3) the convergence of ALADDIN is more robust than those of AR-SGD and SGP to various heterogeneous environments. The main contributions of this work is as follows:

Table 1: Quadrant classification of existing distributed training algorithms (elaborated in Section 2)

	Centralized	**Decentralized**
Synchronous	BSP	AR-SGD
Asynchronous	ASP, SSP, EASGD DSSP, **ALADDIN**	GoSGD, D-PSGD AD-PSGD, SGP

- Identifying the deficiencies of both centralized and decentralized training – i.e., large communication overhead and increased parameter variance problems.
- Proposing a novel asymmetric centralized training algorithm, named as ALADDIN, successfully addressing both problems simultaneously.
- Providing theoretical analysis for the convergence of ALADDIN on non-convex optimization.
- Comprehensive evaluation verifying the effectiveness of ALADDIN in terms of the convergence rate, scalability, and robustness under heterogeneous environments.

2 RELATED WORK

Centralized training. In centralized training, a dedicated PS manages the global parameters by aggregating the training results of workers. In bulk synchronous parallel (BSP) [9, 39], a *synchronous* centralized algorithm, the parameters of all workers are synchronized via PS that aggregates the gradients from all workers at once. Thus, all workers train the model in a consistent direction. However, the synchronization overhead can be significant, especially in a heterogeneous cluster. To reduce the overhead, a lot of asynchronous training algorithms have been studied. In asynchronous parallel (ASP) [25], PS aggregates the gradients from workers in an *asynchronous* manner: PS does not wait for all workers; instead, PS processes the gradients from each worker *individually*. However, the parameter variance among workers could be large if there are workers with different training speeds, causing global model convergence delayed. Stale synchronous parallel (SSP) [12], positioned in the middle of BSP and ASP, relaxes the parameter synchronization by allowing workers to train with different parameter versions. However, SSP controls the difference among the versions to be always less than the pre-defined threshold. Dynamic SSP (DSSP) [38] further improves SSP by varying the threshold dynamically as training progresses. Elastic averaging SGD (EASGD) [37] reduces the communication overhead via periodic communication between PS and workers. The centralized training algorithms often suffer from the problem of PS being a bottleneck because PS aggregates training results from all workers. To mitigate this problem, parameter sharding [5, 15] is generally applied to centralized training algorithms. This technique divides global parameters and distributes them into multiple PSs to process them in parallel.

Decentralized training. In decentralized training, the training results of workers are aggregated via peer-to-peer communication

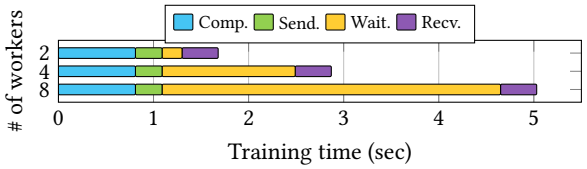


Figure 2: Breakdown of training time with 2, 4, 8 workers for the VGG-16 model in centralized training.

without PS. In AllReduce-SGD (AR-SGD) [10, 27], a synchronous decentralized algorithm, the parameters of all workers are synchronized via AllReduce communication at every iteration. In synchronous training, Goyal et al. [10] uses the linear learning rate scaling with gradual warm-up (LSW) to speed up the synchronous training with large batch size. AdaScale [16] further improves LSW [10] by applying more reliable learning rate considering the variance of gradients. The synchronization overhead, however, can be a big problem like BSP. In decentralized parallel SGD (D-PSGD) [21], each worker exchanges its parameters with another worker and updates its parameters by averaging them at every iteration. D-PSGD proved that the convergence rate of decentralized training is comparable to that of SGD through the theoretical analysis. AD-PSGD [22], the asynchronous version of D-PSGD, improves the training performance of D-PSGD by allowing workers to communicate with each other *asynchronously*. The Stochastic Gradient Push (SGP) [1], a state-of-the-art decentralized training algorithm, adopts the push-sum gossip algorithm [17] to efficiently aggregate parameter aggregation of workers. The Cooperative-SGD [29] proposes a unified framework for generalizing existing distributed training algorithms and provides its convergence analysis within the generalized framework, where fully-synchronous SGD (BSP, AR-SGD), EASGD, and D-PSGD are represented as its special cases. [23] provides a general consistency condition that covers existing asynchronous distributed training algorithms.

3 THE PROPOSED ALGORITHM: ALADDIN

In this work, we consider the following problem:

$$\min_{x^1,\ldots,x^n,\tilde{x}} \sum_{i=1}^{n} \mathbb{E}[F(x^i,\xi^i)] + \frac{\rho}{2}||x^i-\tilde{x}||^2, \tag{1}$$

where n is the number of workers, x^i is the set of local parameters of worker i, ξ^i is the local data of worker i, $F(x^i,\xi^i)$ is the local loss function of the local parameters x^i given data samples ξ^i, and $\tilde{x}$ is the global parameters. The goal of Eq. 1 is twofold: (1) the first term minimizes the loss function (thus maximizing accuracy) and (2) the second term minimizes the variance between local and global parameters (thus preserving the consistency among parameters across workers). ALADDIN achieves both objectives in Eq. 1 and successfully addresses the deficiencies of both centralized and decentralized training.

3.1 Asymmetric Centralized Training

Breakdown of centralized training. As explained in Section 1, our research direction (D1) is *to improve centralized training by reducing the large communication overhead* caused by the PS bottleneck. Prior to developing a new centralized algorithm, we investigate why the PS bottleneck significantly increases the communication overhead in centralized training. We measure both computation and communication times at each training iteration in a centralized training algorithm, ASP (i.e., Hogwild) [25], where communication is divided into three parts: sending, waiting, and receiving. Figure 2 shows the breakdown of the total measured time (computing, sending, waiting, and receiving) in the training of the VGG-16 model. As the number of workers increases, the overall communication time increases proportionally. In particular, the communication time occupies more than 80% of the total training time when there are 8 workers.

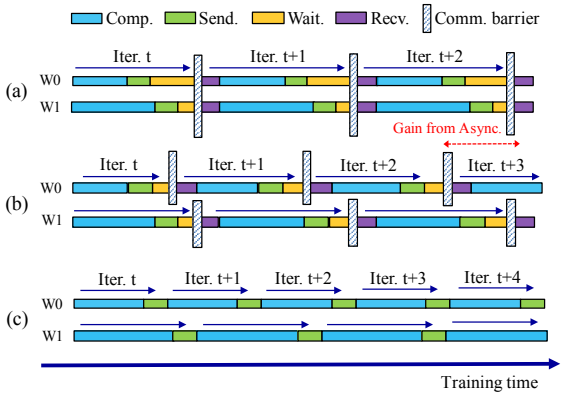


Figure 3: Three types of communication: (a) Synchronous, (b) Asynchronous, and (c) Asymmetric.

Further, we observe that most of the increased communication time comes from the "waiting" time as shown in Figure 2. Looking more closely, in centralized training, a worker communicates with PS as follows: a worker (1) sends its training results (i.e., gradients) to PS, (2) *waits for PS to send back the newly updated global parameters*, (3) replaces its local parameters by the global parameters, and then (4) proceeds to the next iteration. Therefore, workers are not able to proceed to the next iteration while "waiting" for PS in step (2). This result gives us an important lesson: to reduce the overall communication cost in centralized training, it is critical to reduce the idle time of workers, "symmetrically" waiting for the updated global parameters from PS. From this understanding, next, we propose a novel "asymmetric training" between PS and workers.

Effect of asymmetric training. Figure 3 illustrates the difference between our asymmetric training and existing synchronous and asynchronous training. In synchronous communication (Figure 3(a)), the parameters of all workers are synchronized simultaneously at a communication barrier. The synchronization overhead might be quite significant when some workers fall behind other workers (i.e., straggler), which may degrade the training performance significantly. While, asynchronous communication (Figure 3(b)) between PS and workers improves upon synchronous one (Figure 3(a)) such that *a worker does not need to wait for stragglers* to proceed to its next iteration. As shown in Figure 3(b), however, there are still communication barriers at the end of every iteration in asynchronous training due to its "symmetric" communication. We highlight again that this is a critical factor that causes high communication overhead in asynchronous centralized training.

On the other hand, in our asymmetric training, a worker sends its training results to PS and then *immediately proceeds to the next iteration without waiting for PS* as illustrated in Figure 3(c). Since there is no communication barrier, a worker can train a model in a "completely wait-free manner" regardless of whether PS being a bottleneck or not at that time, which means that *workers do not wait for other workers nor PS*. Thus, our asymmetric training can successfully address the problem of large communication overhead caused by the PS bottleneck. From this point of view, existing asynchronous centralized training algorithms such as ASP (i.e., Hogwild) [25], SSP [12], and DSSP [38] can be considered as "partially wait-free."

In naive asymmetric training (Figure 3(c)), however, since workers do not receive the global parameters from PS, the model parameters of workers are *never updated*. Then, it results in degrading the quality of the gradients computed by workers, interfering with the global model convergence. To address this issue, we propose updating strategies for local and global parameters to speed up the model convergence in our asymmetric training.

3.2 Parameter Update for Model Convergence

Updating local parameters. Let us explain the strategy for updating local parameters of workers. The local parameters of each worker are updated with respect to both *local and global aspects*. From the local aspect, first, a worker not only sends the computed gradients to PS asymmetrically, *but also updates its local parameters* (i.e. *local self-update*) using $x^i_{k+1} = x^i_k - \eta \cdot g^i_k$, where x^i_k and g^i_k are the local model and local gradients of worker i at iteration k, respectively, and η is the learning rate. Through this simple *local self-update*, the quality of local gradients can be much improved *without* receiving the global parameters from PS. However, the parameters of workers are trained only *locally*. Therefore, it is likely to increase the parameter variance among workers, which means that the variance of the gradients computed from their local parameters also increases, adversely affecting the global model convergence.

For this reason, from the global aspect, second, we update local parameters of workers by *exploiting the up-to-date global parameters*, sent from PS in every π iterations at its own decision (i.e., *PS-triggered local update*). In this way, the local parameters of all workers could be mixed indirectly through the global parameters, which reduces the parameter variance among workers. Specifically, inspired by Zhang et al. [37], we define the update rule for local parameters of each worker x^i, based on the problem formulation represented in Eq. 1. By taking the gradient descent with respect to x^i on Eq. 1, we get:

$$x^i = x^i - \eta \cdot (g^i + \rho(x^i - \tilde{x})) \tag{2}$$

Denote $\alpha = \eta \cdot \rho$ and $x^i = x^i - \eta \cdot g^i_t$ which can be considered equal to the local parameter update of worker i. Then, we get the following update rule for x^i:

$$x^i = x^i - \alpha(x^i - \tilde{x}) = (1 - \alpha)x^i + \alpha\tilde{x} \tag{3}$$

Eq. 3 implies that a worker updates its local parameters by taking the weighted average between its local parameters x^i and the global parameters $\tilde{x}$ where α is an oscillating weight factor between local and global parameters. We note that this formulation can generalize diverse parameter update rules in distributed training. For example, if $\alpha = 0$, it is the same as local training only (i.e., ensemble), while, if $\alpha = 1$, it degenerates to asynchronous centralized training algorithms (e.g., Hogwild [25]) where local parameters of each worker are completely replaced by the global parameters.

Algorithm 1 Training of worker i in ALADDIN

Require: Initialize x^i_0 for worker $i \in \{1, 2, ..., n\}$, dataset ξ^i, batch size B, learning rate η

1: **for** $t = 0, 1, \ldots$ **do**
2: $\quad g^i_t \leftarrow \frac{1}{B} \sum_{j=1}^{B} \nabla F(x^i, \xi^i_{t,j})$
3: $\quad x^i_{t+1} \leftarrow x^i_t - \eta \cdot g^i_t$ (*Local self-update*)
4: $\quad$ Send g^i_t to PS
5: $\quad$ **if** $\tilde{x}, \alpha_i$ **from PS**
6: $\quad\quad x^i \leftarrow (1 - \alpha_i)x^i + \alpha_i\tilde{x}$ (*PS-triggered local update*)
7: $\quad$ **end if**
8: **end for**

We set the weight α based on the following intuition. The less the amount of the training results of a worker is applied to the global parameters, the larger the distance between the local parameters of the worker and the global parameters becomes. By this intuition, the variance may increase more if the number of workers n and the update period π get larger. Thus, we set $\alpha = 1 - \frac{1}{\sqrt{n\pi}}$ to increase α with the increasing number of workers n and update period π, helping local parameters of workers to *quickly catch up the global parameters*. Thus, the weight of global parameters α is automatically decided by hyperparameter π and the number of workers n.

As we explained, hyperparameter π controls the period for PS-triggered local update of each worker. When π is large, each worker trains its model in many iterations locally without receiving the global parameters from PS. Thus, the training throughput of each worker per unit time increases, while the parameter variance among workers also increases, which may adversely affects the model convergence. On the other hand, when π is small, each worker receives the global parameters from PS frequently, which can degrade the overall training performance of ALADDIN. Therefore, the model convergence and training performance of ALADDIN are in a *trade-off* relationship according to the value of π. We will empirically show the impact of π on model convergence and training performance of ALADDIN in Section 5.6. The whole training process of a worker is described in Algorithm 1. Lines 3 and 6 represent the local parameter updates in local and global aspects, respectively.

Our strategy for updating local parameters has several advantages in terms of model convergence and training performance, compared to existing work [1, 22, 29, 37]. First, as explained in Section 1, since ALADDIN belongs to centralized training, the parameter variance among workers could be smaller than that in decentralized training if *the global parameters are directly exploited*. Second, unlike [29, 37] where workers communicate with PS (or other workers) only when they reach the end of an update period, workers *send the gradients to the PS at every iteration* even if they do not reach the update period (i.e., asymmetric training). That is, *the gradients most recently computed* from workers are included in the global parameters. Thus, when a worker updates the local parameters in the global aspect (i.e., the PS-triggered local update), *the latest information of all other workers are applied to its local parameters*, helping reduce further parameter variance among workers. Third,

Algorithm 2 Training of PS in ALADDIN

Require: Global model $\tilde{x}_0$, learning rate η, update period π, period count c_i for workers

1: $\tilde{c} \leftarrow 0, c_i \leftarrow 0$
2: **for** $t = 0, 1, \ldots$ **do**
3: Receive g^i from worker i
4: $\tilde{c} \leftarrow \tilde{c} + 1, c_i \leftarrow c_i + 1$
5: **if** $c_i < \pi$
6: $\tilde{g} \leftarrow \tilde{g} + g^i$ (*Gradient accumulation*)
7: **else**
8: $\tilde{x} \leftarrow \tilde{x} - \eta \cdot \tilde{g}, \ \tilde{g} \leftarrow \frac{1}{\tilde{c}} \cdot \tilde{g}$ (*Lazy global update*)
9: $\alpha_i \leftarrow 1 - \frac{1}{\sqrt{\tilde{c}}}$
10: Send $\tilde{x}$ to worker i (*PS-triggered local update*)
11: $\tilde{g} \leftarrow \emptyset, \tilde{c} \leftarrow 0, c_i \leftarrow 0$
12: **end if**
13: **end for**

in our PS-triggered local update, PS sends the global parameters to workers *independently* while workers are training their models. As a result, workers *do not wait for PS*, in order to receive global parameters (i.e., no communication barrier). Thus, all workers can train their models in a "completely wait-free" manner.

Updating global parameters. In our asymmetric training, PS sends the global parameters to each worker only once in every π iterations, rather than whenever PS receives local gradients from a worker. As such, *PS can postpone applying the gradients from workers to the global parameters until the PS-triggered local update is required without any loss*. For this reason, we propose a *lazy* updating strategy for global parameters (*Lazy global update*). In this strategy, PS just accumulates the gradients from workers without applying them to the global parameters, and updates the global parameters only when the PS-triggered local update is required. The whole training process of PS is described in Algorithm 2. Lines 8 and 9 represent lazy global update for the global parameters and PS-triggered local update for local parameters, respectively.

This global update strategy also has advantages in terms of model convergence and training performance, compared to existing asynchronous centralized training algorithms [6, 12, 19, 25, 36, 38] that adopt *symmetric* communication, where PS receives gradients from a worker and reflects them in the global parameters *individually* for that worker. First, our strategy can reduce the number of updating operations considerably such as $apply\_gradient(\cdot)$ in Tensorflow (thus reducing computational cost of PS), compared to those of existing algorithms. This is because our strategy applies the accumulated gradients to the global parameters only when the PS-triggered local update is necessary. It means that the computing resources of PS are used more efficiently in ALADDIN. Large batch training [10, 16, 32–34], whose goal is to speed up SGD, increases the size of data samples processed in each iteration (i.e., batch size), in order to reduce the variance of stochastic gradient. With the same principle, second, our lazy global update also *reduces the variance of gradients by PS aggregating gradients from workers*, which makes the training of the global parameters in PS more reliable. We will demonstrate the effectiveness of our updating strategies for local and global parameters in Section 5.

3.3 Parameter Sharding

To effectively manage the communication overhead of a parameter server (PS) in ALADDIN, we carefully apply the technique of parameter sharding to ALADDIN. Parameter sharding [5, 6, 15] divides global parameters and distributes them into multiple PSs to process them in parallel. *Layer-wise parameter sharing* is normally used because a layer is represented as a single data structure in many DL frameworks (e.g., Tensor in Tensorflow) so that it can be processed sequentially. In some DNN models, however, a single layer may contain the majority of the entire model parameters. For instance, in the VGG-16 model, the last two fully-connected layers are in charge of more than 90% of all parameters. In this case, the PS in charge of the extremely large layers suffers from a big burden, thereby becoming a bottleneck in the overall training. To address this issue, we apply *memory-wise parameter sharding* to our algorithm. Memory-wise sharding defines the maximum size of sharded parameters and has multiple PSs process the parameters in a layer together if it is larger than the maximum size.

4 CONVERGENCE ANALYSIS

In this section, we show that the model trained by ALADDIN converge at a consistent rate with that of SGD and existing state-of-the-art algorithms [1, 22, 29] on non-convex optimization. Our analysis is based on [29] that proves the averaged model of all workers converges when the models of workers are updated by Eq. 4.

$$X_{k+1} = (X_k - \eta \cdot G_k) \cdot W_k, X_k = [x_k^1, \cdots, x_k^n], G_k = [g_k^1, \cdots, g_k^n] \quad (4)$$

where x_k^i is the local model of worker i at iteration k, g_k^i is i's gradients at iteration k, and η is the learning rate. W_k is an $n \times n$ *mixing matrix* representing the relationship among workers, consisting of w_{ij} that is the $(i, j)^{th}$ element of the mixing matrix W_k representing the weight of worker i in the averaged model at worker j. To guarantee the model convergence, W_k has to satisfy that (1) it is *doubly-stochastic* and (2) the magnitude of its second largest eigenvalue is less than 1 (i.e., *spectral gap*) [29].

We will prove the convergence of ALADDIN by showing that W_k for ALADDIN satisfies the two conditions above. [1] We note again that [29] shows that the *averaged model* of workers converges. On the other hand, since ALADDIN is a centralized algorithm where a global model is maintained physically in PS, we need to prove the convergence of the *global model* at PS (not the averaged model). To this end, we first show that in ALADDIN, *the global model is equivalent to the averaged model of all workers*.

Lemma 1 *Let x_k^i and $\tilde{x}_k$ be the local model of worker i and the global model at iteration k in ALADDIN, respectively. Then, the averaged model of all workers is equivalent to the global model.*

$$\tilde{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_k^i \quad (5)$$

In order to exploit the analysis results of [29], we need to represent the training of ALADDIN as the same form as Eq. 4. However, since the dimension of W_k is $n \times n$ (thus only representing relationships among workers not PS), aggregating (or mixing) local models of workers via PS cannot be represented. Thus, we show that there

[1] All proofs of lemmas 1, 2 and theorem 1 are included in the following anonymous link: https://sites.google.com/view/aladdin-proofs

exists W_k that results in the same consequence as ALADDIN and that W_k satisfies the two required conditions.

Lemma 2 *Let W_k be an $n \times n$ mixing matrix and α be the weight of the global model in the PS-triggered local update. Then, there exists W_k causing the same consequence as the training of ALADDIN.*

$$W_k = \begin{bmatrix} 1-\frac{\alpha}{n}(n-1) & \frac{\alpha}{n} & \cdots & \frac{\alpha}{n} \\ \frac{\alpha}{n} & 1-\frac{\alpha}{n}(n-1) & \cdots & \frac{\alpha}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha}{n} & \frac{\alpha}{n} & \cdots & 1-\frac{\alpha}{n}(n-1) \end{bmatrix} \quad (6)$$

Note that W_k in Eq. 6 is clearly doubly-stochastic, that is, the sum of each column/raw is 1. Besides, the spectral gap condition is also satisfied as all workers are connected to each other [13]. Now, by lemmas 1 and 2, we are ready to use the analysis results of [29].

Theorem 1 (Convergence of ALADDIN) *Let L, σ^2, ζ, π, and $\tilde{x}^*$ be the Lipschitz constant, variance bound for gradients, magnitude of second largest eigenvalue, update period, and optimal model, respectively. Then, the convergence rate of ALADDIN is as follows.*

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\|\nabla f(\tilde{x}_k)\|^2 \leq \frac{2[f(\tilde{x}_0)-f(\tilde{x}_*)]}{\eta K} + \frac{\eta L\sigma^2}{n} + \eta^2 L^2\sigma^2\left(\frac{1+\zeta^2}{1-\zeta^2}\pi - 1\right) \quad (7)$$

Remark 1 (Consistency with SGD) Despite the asymmetric communication between PS and workers, the convergence rate of ALADDIN is consistent with those of SGD and existing work [22, 29].

Corollary 1 With a proper learning rate $\eta = \frac{1}{L}\sqrt{\frac{n}{K}}$,

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\|\nabla f(\tilde{x}_k)\|^2 \leq \frac{2L[f(\tilde{x}_0)-f(\tilde{x}_*)] + \sigma^2}{\sqrt{nK}} + \frac{n\sigma^2}{K}\left(\frac{1+\zeta^2}{1-\zeta^2}\pi - 1\right) \quad (8)$$

Remark 2 (Linear speedup) In the right side of Eq. 8, the first term dominates the second term if the total number of iterations K is sufficiently large. Thus, the convergence rate of ALADDIN is $O(\frac{1}{\sqrt{nK}})$, which means that the convergence of ALADDIN is accelerated linearly as the number of workers increases.

5 EXPERIMENTAL VALIDATION

In this section, we evaluate ALADDIN by answering the following five research questions:

- RQ1. Does ALADDIN provide the convergence rate w.r.t. time and epochs higher than existing algorithms?
- RQ2. Does ALADDIN provide the scalability w.r.t. the number of workers better than existing algorithms?
- RQ3. How robust is ALADDIN to heterogeneous environments more than existing algorithms?
- RQ4. How effective are the updating strategies of ALADDIN in model convergence?
- RQ5. How sensitive are the model convergence and scalability of ALADDIN to the hyperparameter π?

5.1 Experimental Setup

Datasets and models. We evaluate ALADDIN with two widely used CNN models, ResNet-50 [11] and VGG-16 [28]. ResNet-50 with 23M parameters is a *computation-intensive* model, while VGG-16 with 138M parameters is a *communication-intensive* model. As the training datasets for both models, we use CIFAR-10 [18] [2] [18] and ImageNet-1K [26] datasets. [3] [26] datasets. CIFAR-10 consists of 50K training images and 10K test images with 10 labels and ImageNet-1K consists of about 1.2M training images and 5K test images with 1K labels.

System configuration. We use TensorFlow 1.15 and MPICH 3.1.4 to implement all algorithms including ALADDIN on Ubuntu 18.04 OS. We evaluate ALADDIN on the cluster with four machines, where each machine has two NVIDIA RTX 2080 Ti GPUs and an Intel i7-9700k CPU with 64 GB memory. All machines are interconnected by 10Gbps Ethernet (Mellanox ConnectX-4Lx).

Algorithms. We compare ALADDIN with two centralized algorithms, ASP (i.e., Hogwild) [25] and EASGD [37], and two decentralized algorithms, AR-SGD [10] and the state-of-the-art decentralized algorithm SGP [1]. We selected ASP and EASGD employing symmetric communication between PS and workers, to compare the effect of the asymmetric training of ALADDIN. We also selected SGP for a decentralized algorithm because it is reported that SGP outperforms D-PSGD and AD-PSGD in [1]. AR-SGD, a synchronous algorithm, is used as the baseline. For EASGD, we set a moving average rate as $\frac{0.9}{n}$ and a communication period as 2 as recommended in [37]. We use 1-overlap SGP with the directed exponential graph as recommended in [1]. For ALADDIN, we empirically found the best values for the update period π, and set π as 4 for both models. The experimental results about the effects of hyerparameter π on model convergence and training performance of ALADDIN are included in Section 5.6. We note that the weight of global parameters α is decided automatically by hyperparameter π and the number of workers n, as explained in Section 3.2 (i.e., $\alpha = 1 - \frac{1}{\sqrt{n\pi}}$). For a fair evaluation, we apply the memory-wise parameter sharding (Section 3.3) to *all* centralized algorithms (i,e., ASP, EASGD, and ALADDIN) in all experiments.

Hyperparameter setting. We set batch size B as 128 for ResNet-50 and 96 for VGG-16 to fully utilize the GPU memory. We use momentum SGD and set momentum as 0.9, weight decay factor as 0.0001, and learning rate η as $0.01 \times n$ for CIFAR-10 and $0.05 \times n$ for ImageNet-1K, based on the learning rate scaling rule [10]. We apply the learning rate warm-up for the first 20 epochs for CIFAR-10 and first 5 epochs for ImageNet-1K, and decay η by $\frac{1}{10}$ at epoch 150 for CIFAR-10 and epoch 30, 60, and 80 for ImageNet-1K [10].

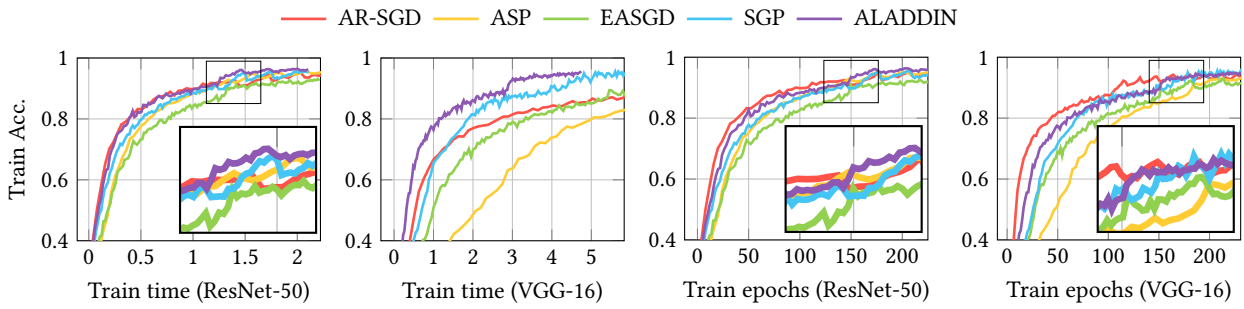
5.2 Convergence Rate

First, we evaluate the model accuracies of all algorithms and their convergence rate with respect to training epochs and time. We compare ALADDIN with other algorithms using the following four metrics: the final test accuracy (Test Acc.), the total training time (Train time), the test accuracy achieved in the given time (Test Acc. (x hrs.)), and time to achieve the given test accuracy (Train time

[2] https://www.cs.toronto.edu/~kriz/cifar.html
[3] http://www.image-net.org/

Table 2: Results for ResNet-50 and VGG-16 on CIFAR-10 and ImageNet-1K.

	ResNet-50 (CIFAR-10)				VGG-16 (CIFAR-10)				ResNet-50 (ImageNet-1k)			
	Test Acc.	Train time	Test Acc. (1 hrs.)	Train time (90%)	Test Acc.	Train time	Test Acc. (2 hrs.)	Train time (90%)	Test Acc.	Train time	Test Acc. (5 hrs.)	Train time (60%)
AR-SGD	0.9353	2.37 hrs.	0.8992	0.98 hrs.	**0.9238**	14.46 hrs.	0.7765	6.40 hrs.	**0.7514**	20.72 hrs.	0.5344	6.83 hrs.
ASP	0.9315	2.24 hrs.	0.8810	1.16 hrs.	0.9171	10.96 hrs.	0.4973	8.22 hrs.	0.7508	18.82 hrs.	0.4851	7.45 hrs.
EASGD	0.9112	2.21 hrs.	0.8431	1.36 hrs.	0.9124	9.04 hrs.	0.7002	6.18 hrs.	0.7012	17.53 hrs.	0.5589	6.05 hrs.
SGP	0.9340	2.11 hrs.	0.8839	1.15 hrs.	0.9228	6.66 hrs.	0.8247	3.94 hrs.	0.7513	17.42 hrs.	0.5545	6.53 hrs.
ALADDIN	**0.9378**	**1.81** hrs.	**0.9071**	**0.92** hrs.	0.9226	**4.97** hrs.	**0.8687**	**2.87** hrs.	0.7508	**13.98** hrs.	**0.6561**	**4.62** hrs.



(a) Time-wise view

(b) Epoch-wise view

Figure 4: Training accuracies w.r.t training time (hrs) and epochs for ResNet-50 and VGG-16 on CIFAR-10.

(x%)). Table 2 shows the results of the training for ResNet-50 and VGG-16 on CIFAR-10 (250 epochs) and ImageNet-1K (90 epochs).

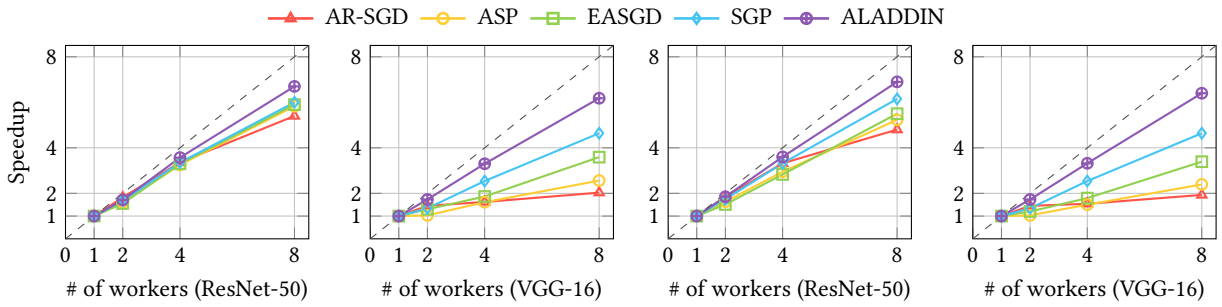
The results demonstrate that (1) *ALADDIN finishes its training in shorter time than all competing algorithms for both models on both datasets*, by 16% to 191%, and (2) *the models trained by ALADDIN converge to high accuracies comparable to or even better than those of the synchronous algorithm (AR-SGD), the ground truth*. In particular, ALADDIN always achieves the highest accuracy in the given time and also achieves the given test accuracy in the shortest time. This result indicates that ALADDIN could be a good choice as a distributed training algorithm in the time-constraint circumstance. ALADDIN always outperforms SGP [1], the state-of-the-art algorithm, in the total training time and time-constraint metrics. This result validates that centralized training could be made to have training efficiency higher than decentralized training if the PS bottleneck problem is addressed successfully.

For more in-depth evaluation for model convergence of each algorithm, we also compare the convergence rates of all algorithms with respect to training time and epochs. Figure 4 shows the results on the training of ResNet-50 and VGG-16 on CIFAR-10. ALADDIN outperforms all competing algorithms in terms of the time-wise convergence rate as shown in Figure 4(a), which is more clearly shown in the results of the VGG-16 training. This result indicates that ALADDIN could achieve higher accuracy than other competing algorithms in the given time. AR-SGD, a synchronous algorithm, shows the best result in the epoch-wise convergence rate as shown in Figure 4(b). This is because all workers always have the same parameters through its parameter synchronization. Thus, they can train their models in the exactly same direction. ALADDIN shows the second-best result in the epoch-wise convergence rate. ALADDIN lags behind AR-SGD in the initial phase of the training, but eventually catches up to AR-SGD. This result is promising, considering ALADDIN finishes the training in shorter time up to 191% than AR-SGD. As a result, ALADDIN has a good convergence rate with respect to both the training time and epochs, which verifies that ALADDIN successfully addresses the limitations of both centralized and decentralized training – large communication overhead and high parameter variance problems – simultaneously as we claimed.

5.3 Scalability

As the number of workers in a distributed cluster increases, the communication overhead required in distributed training inevitably increases as well. With the increase of the communication overhead, the training throughput per unit time of each worker decreases, which may adversely affect the scalability of distributed training. Thus, in this experiment, we evaluate the scalability of each distributed training algorithm with the increasing number of workers. We measure the training throughput of each algorithm as the number of workers increases, in the ResNet-50 and VGG-16 training on CIFAR-10 and ImageNet-1K. Figure 5 shows the results.

ALADDIN provides the best speed-up results for both models and both datasets. In the computation-intensive ResNet-50 training (thus low communication overhead), all algorithms show good speed-up results with the increasing number of workers. However, in the communication-intensive training with VGG-16 (thus high communication overhead), both AR-SGD and ASP show poor results. These results indicate that the synchronization overhead of AR-SGD is significant in the training of a communication-intensive model, and the symmetric communication between PS and workers in ASP causes workers to wait for PS as we claimed in Section 3.1, which negatively affects the scaling up the training throughput with the increasing number of workers.



(a) CIFAR-10 (b) ImageNet-1K

Figure 5: Speed-up results for ResNet-50 and VGG-16 on CIFAR-10 and ImageNet-1K with the increasing number of workers.

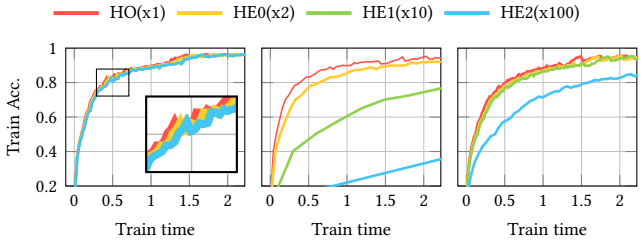
SGP, the state-of-the-art decentralized algorithm [1], shows better speed-up results than other algorithms except for ALADDIN in training with both models. This is because SGP aggregates the training results of workers in a decentralized manner using the push-sum gossip algorithm [17]. ALADDIN provides speed-up results up to 34% better than SGP. This is surprising result, considering that ALADDIN is a centralized training algorithm that has the PS bottleneck problem in its nature. Therefore, we conclude that our asymmetric training successfully addresses the problem of high communication overhead by the PS bottleneck problem.

5.4 Robustness to Heterogeneous Environment

In a heterogeneous environment, there are workers with varying training speeds, which may adversely affect the model convergence and training performance of distributed training. For example, the parameter variance among workers may be increased by slow workers, which leads to delayed model convergence. Also, the training performance can be degraded since slow workers cause other workers to wait at communication barriers. Thus, distributed training algorithms should be evaluated in this aspect. To evaluate ALADDIN in this aspect, we configure one homogeneous and three heterogeneous clusters: (0) **HO**: a homogeneous cluster with no slow worker, (1) **HE0**: a heterogeneous cluster with a ×2 slower worker, (2) **HE1**: a heterogeneous cluster with a ×10 slower worker, and (3) **HE2**: a heterogeneous cluster with a ×100 slower worker.

We train the ResNet-50 model using AR-SGD, SGP, and ALADDIN on the four clusters, and measure their accuracies with respect to the training time. We also measure the speed-up results of the algorithms when there are 8 workers in the clusters. Figure 6 and Table 3 show the results. Clearly, *ALADDIN is most robust to all heterogeneous clusters in terms of both the model convergence and training performance*. In particular, regardless of the degree of heterogeneity, ALADDIN always shows a consistent convergence rate and best speed-up results. Note that the small loss is inevitable because of the slow worker. This result indicates that ALADDIN effectively reduces the parameter variance increased by slow workers, and removes communication barriers in distributed training on heterogeneous environments.

AR-SGD shows significant degradation in its convergence rate. This is because AR-SGD is a *synchronous* algorithm such that workers do not proceed to the next iteration until the slowest worker finishes its gradient computation at each iteration. The convergence of SGP is relatively robust to heterogeneous environments, but the loss is quite large in the **HE2** setting with an extremely (×100) slower worker. This result means that in spite of the decentralized communication, a very slow worker may slow down the entire training since the delay from the slow worker is propagated to other workers step by step via peer-to-peer communication.



(a)ALADDIN (b) AR-SGD (c) SGP

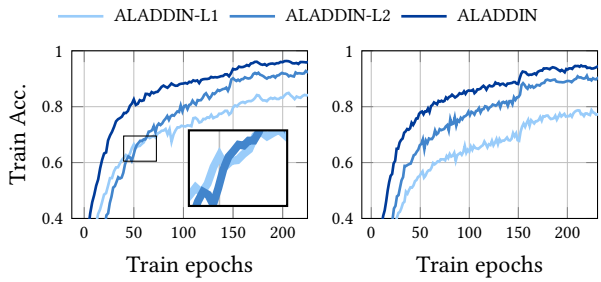
Figure 6: Accuracies w.r.t train time on hetero clusters.

Table 3: Speed-up results on heterogeneous clusters.

	ALADDIN		AR-SGD		SGP	
	ResNet	VGG	ResNet	VGG	ResNet	VGG
HO (×1)	**6.70**	**6.18**	5.40	2.03	6.00	4.64
HE0 (×2)	**6.30**	**5.74**	3.22	1.46	5.21	4.15
HE1 (×10)	**5.95**	**5.49**	0.71	0.62	4.83	3.81
HE2 (×100)	**5.87**	**5.41**	0.08	0.07	1.93	1.89

5.5 Ablation Study: Updating Strategies

In this experiment, we verify the effectiveness of each updating strategy (Section 3.2) on the model convergence of ALADDIN. We compare the following three versions of ALADDIN: (1) ALADDIN-L1 is asymmetric training with the local self-update, (2) ALADDIN-L2 is asymmetric training with the local self-update and the PS-triggered local update, and (3) ALADDIN is with all strategies (i.e., the local self-update, PS-triggered local update, and lazy global update). We train ResNet-50 and VGG-16 models on CIFAR-10 using three versions of ALADDIN and measure their accuracies with respect to training epochs. As shown in Figure 7, ALADDIN converges to the highest accuracy, while ALADDIN-L1 converges to the lowest accuracy in both models. These results demonstrate that *each of our updating strategies is effective in improving the convergence of the model trained by ALADDIN*.



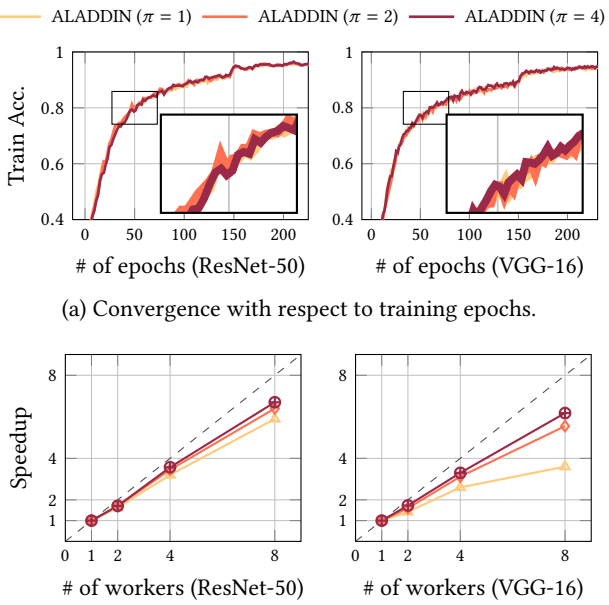
(a) ResNet-50 (b) VGG-16

Figure 7: Ablation study: effects of the updating strategies of ALADDIN on model convergence.

Let us analyze the results more deeply. Interestingly, in the ResNet-50 training, ALADDIN-L1 achieves higher accuracy than ALADDIN-L2 before around 50 epochs as shown in Figure 7(a). This is because mixing the parameters of workers via the PS-triggered local update rather interferes with the global model convergence in the beginning of training, where the model of each worker tends to fluctuate steeply. On the other hand, in ALADDIN-L1 with only the local self-update, each worker can train its model alone without any interference from other workers. But, ALADDIN-L2 achieves much higher accuracy than ALADDIN-L1 in the end. This result means that, in the training of ALADDIN-L1, the parameter variance among workers is getting higher and adversely affects the model convergence eventually. Lastly, ALADDIN always shows the best accuracy across all training iterations. This result demonstrates that the lazy global update successfully reduces the variance of gradients, making the training more reliable even in the beginning of training as we claimed in Section 3.2. Through this ablation study, we verify that our strategies for updating local and global parameters help to speed up the convergence of ALADDIN.

5.6 Hyperparameter Sensitivity: π

As explained in Section 3.2, hyperparameter π controls the period of PS-triggered local update of each worker. As π gets larger, the training performance increases since each worker trains its model in more iterations locally without receiving the global parameters from PS. However, its model convergence may be delayed as the parameter variance among workers increases as well. Thus, both model convergence and training performance of ALADDIN are in tension per π. In this experiment, we evaluate the impact of the period π on the model convergence and training performance of ALADDIN. We compare the accuracies and scalability of ALADDIN with varying $\pi = 1, 2, 4$ in the training of ResNet-50 and VGG-16 on CIFAR-10. As shown in Figure 8(a), *the convergence of ALADDIN is insensitive to π, where the model trained by ALADDIN converges at a consistent rate regardless of π*. This result indirectly verifies that the updating strategies of ALADDIN effectively reduce the parameter variance among workers, despite the large update period π. Also, ALADDIN shows *almost linear speed-ups* as the number of workers increases, except for the VGG-16 training with $\pi = 1$ (Figure 8(b)). Considering these results of the model convergence and speed-up, according to π, we conclude that ALADDIN is not hyperparameter-sensitive, achieving almost a linear speed-up with the increasing number of workers, while maintaining high accuracy.



(a) Convergence with respect to training epochs.

(b) Speed-up results with the increasing number of worekrs.

Figure 8: Sensitivity of model convergence and scalability to hyperparameter π.

6 CONCLUSION

This paper identified the deficiencies of both centralized and decentralized training – i.e., large communication overhead and high parameter variance, and proposed a novel asymmetric centralized training algorithm, named as ALADDIN that effectively resolves both problems at the same time. We provided the theoretical analysis for the convergence of ALADDIN, where the model trained by ALADDIN converges at a consistent rate, $O(\frac{1}{\sqrt{nk}})$, being comparable to those of existing state-of-the-art works. Through comprehensive experiments, we showed that models trained by ALADDIN converge to accuracies, comparable to those of AR-SGD, a synchronous algorithm, within the shortest time, and ALADDIN provides almost linear speed-up as the number of workers increases in both computation-intensive and communication-intensive models. Further, via the experiments on three heterogeneous environments, we demonstrated that ALADDIN is robust to heterogeneous environments more than existing state-of-the-art algorithms, where the models trained by ALADDIN converge at a consistent rate regardless of the degree of heterogeneity. Finally, we also verified that each of our updating strategies is effective in speeding up the model convergence of ALADDIN, and the convergence and scalability of ALADDIN is insensitive to the hyperparameter π.

ACKNOWLEDGMENTS

The work of Sang-Wook Kim was supported by the National Research Foundation of Korea (NRF) under Project Number NRF-2020R1A2B5B03001960, and Institute of Information Communications Technology Planning Evaluation (IITP) under Project Number 2020-0-01373 (Artificial Intelligence Graduate School Program, Hanyang University). The work of Dongwon Lee was supported by the NSF award #212114824.

REFERENCES

[1] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. 2019. Stochastic gradient push for distributed deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 344–353.

[2] Michael Blot, David Picard, and Matthieu Cord. 2018. GoSGD: Distributed Optimization for Deep Learning with Gossip Exchange. *arXiv preprint arXiv:1804.01852* (2018).

[3] Michael Blot, David Picard, Matthieu Cord, and Nicolas Thome. 2016. Gossip training for deep learning. *Proceedings of the Advances in Neural Information Processing Systems Workshop on Optimization for Machine Learning* (2016).

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[5] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*). 571–582.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Proceedings of the Advances in Neural Information Processing Systems*. 1223–1231.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Nima Eshraghi and Ben Liang. 2020. Distributed Online Optimization over a Heterogeneous Network with Any-Batch Mirror Descent. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 2933–2942.

[9] Alexandros V Gerbessiotis and Leslie G Valiant. 1994. Direct bulk-synchronous parallel algorithms. *J. Parallel and Distrib. Comput.* 22, 2 (1994), 251–267.

[10] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[12] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Proceedings of the Advances in Neural Information Processing Systems*. 1223–1231.

[13] Shlomo Hoory, Nathan Linial, and Avi Wigderson. 2006. Expander graphs and their applications. *Bull. Amer. Math. Soc.* 43, 4 (2006), 439–561.

[14] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the Advances in Neural Information Processing Systems*. 103–112.

[15] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware distributed parameter servers. In *Proceedings of the ACM International Conference on Management of Data, (SIGMOD)*. 463–478.

[16] Tyler Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. 2020. AdaScale SGD: A User-Friendly Algorithm for Distributed Training. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 4911–4920.

[17] David Kempe, Alin Dobra, and Johannes Gehrke. 2003. Gossip-based computation of aggregate information. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*. 482–491.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[19] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*. 583–598.

[20] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. 2018. Pipe-SGD: a decentralized pipelined SGD framework for distributed deep net training. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*. 8056–8067.

[21] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems*. 5330–5340.

[22] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2018. Asynchronous decentralized parallel stochastic gradient descent. In *Proceedings of the international Conference on Machine Learning (ICML)*. 3049–3058.

[23] Giorgi Nadiradze, Ilia Markov, Bapi Chatterjee, Vyacheslav Kungurtsev, and Dan Alistarh. 2021. Elastic Consistency: A Practical Consistency Model for Distributed Stochastic Gradient Descent. (2021).

[24] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 1–15.

[25] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*. 693–701.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[27] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[28] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[29] Jianyu Wang and Gauri Joshi. 2018. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *arXiv preprint arXiv:1808.07576* (2018).

[30] Shaoqi Wang, Aidi Pi, and Xiaobo Zhou. 2019. Scalable distributed dl training: Batching communication and computation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 33. 5289–5296.

[31] Ran Xin, Soummya Kar, and Usman A Khan. 2020. Decentralized stochastic optimization and machine learning: A unified variance-reduction framework for robust performance and fast convergence. *IEEE Signal Processing Magazine* 37, 3 (2020), 102–113.

[32] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888* (2017).

[33] Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large-batch training for LSTM and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–16.

[34] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962* (2019).

[35] Chen Yu, Hanlin Tang, Cedric Renggli, Simon Kassing, Ankit Singla, Dan Alistarh, Ce Zhang, and Ji Liu. 2019. Distributed learning over unreliable networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 7202–7212.

[36] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. 2017. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 181–193.

[37] Sixin Zhang, Anna E Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Proceedings of the Advances in Neural Information Processing Systems*. 685–693.

[38] Xing Zhao, Aijun An, Junfeng Liu, and Bao Xin Chen. 2019. Dynamic stale synchronous parallel distributed training for deep learning. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1507–1517.

[39] Xing Zhao, Manos Papagelis, Aijun An, Bao Xin Chen, Junfeng Liu, and Yonggang Hu. 2019. Elastic Bulk Synchronous Parallel Model for Distributed Deep Learning. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. IEEE, 1504–1509.

[40] Zhengyuan Zhou, Panayotis Mertikopoulos, Nicholas Bambos, Peter Glynn, Yinyu Ye, Li-Jia Li, and Li Fei-Fei. 2018. Distributed Asynchronous Optimization with Unbounded Delays: How Slow Can You Go?. In *Proceedings of the International Conference on Machine Learning (ICML)*. 5970–5979.